XmlSerializer Class

Namespace: System.Xml.Serialization

Assembly: System.Xml.XmlSerializer.dll

Serializes and deserializes objects into and from XML documents. The XmlSerializer enables you to control how objects are encoded into XML.

In this article

Definition

Examples

Remarks

Constructors

Methods

Events

Applies to

Thread Safety

See also

C# Copiar

```
public class XmlSerializer
```

Inheritance Object → XmlSerializer

Examples

The following example contains two main classes: `PurchaseOrder` and `Test`. The `PurchaseOrder` class contains information about a single purchase. The `Test` class contains the methods that create the purchase order, and that read the created purchase order.

C# Copiar

```
using System;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
```

```
/* The XmlRootAttribute allows you to set an alternate name
   (PurchaseOrder) of the XML element, the element namespace; by
   default, the XmlSerializer uses the class name. The attribute
   also allows you to set the XML namespace for the element.  Lastly,
   the attribute sets the IsNullable property, which specifies whether
   the xsi:null attribute appears if the class instance is set to
   a null reference. */
[XmlRootAttribute("PurchaseOrder", Namespace="http://www.cpandl.com",
IsNullable = false)]
public class PurchaseOrder
{
   public Address ShipTo;
   public string OrderDate;
   /* The XmlArrayAttribute changes the XML element name
    from the default of "OrderedItems" to "Items". */
   [XmlArrayAttribute("Items")]
   public OrderedItem[] OrderedItems;
   public decimal SubTotal;
   public decimal ShipCost;
   public decimal TotalCost;
}

public class Address
{
   /* The XmlAttribute instructs the XmlSerializer to serialize the Name
      field as an XML attribute instead of an XML element (the default
      behavior). */
   [XmlAttribute]
   public string Name;
   public string Line1;

   /* Setting the IsNullable property to false instructs the
      XmlSerializer that the XML attribute will not appear if
      the City field is set to a null reference. */
   [XmlElementAttribute(IsNullable = false)]
   public string City;
   public string State;
   public string Zip;
}

public class OrderedItem
{
   public string ItemName;
   public string Description;
   public decimal UnitPrice;
   public int Quantity;
   public decimal LineTotal;

   /* Calculate is a custom method that calculates the price per item,
      and stores the value in a field. */
```

```
    public void Calculate()
    {
       LineTotal = UnitPrice * Quantity;
    }
 }

 public class Test
 {
    public static void Main()
    {
       // Read and write purchase orders.
       Test t = new Test();
       t.CreatePO("po.xml");
       t.ReadPO("po.xml");
    }

    private void CreatePO(string filename)
    {
       // Create an instance of the XmlSerializer class;
       // specify the type of object to serialize.
       XmlSerializer serializer =
       new XmlSerializer(typeof(PurchaseOrder));
       TextWriter writer = new StreamWriter(filename);
       PurchaseOrder po=new PurchaseOrder();

       // Create an address to ship and bill to.
       Address billAddress = new Address();
       billAddress.Name = "Teresa Atkinson";
       billAddress.Line1 = "1 Main St.";
       billAddress.City = "AnyTown";
       billAddress.State = "WA";
       billAddress.Zip = "00000";
       // Set ShipTo and BillTo to the same addressee.
       po.ShipTo = billAddress;
       po.OrderDate = System.DateTime.Now.ToLongDateString();

       // Create an OrderedItem object.
       OrderedItem i1 = new OrderedItem();
       i1.ItemName = "Widget S";
       i1.Description = "Small widget";
       i1.UnitPrice = (decimal) 5.23;
       i1.Quantity = 3;
       i1.Calculate();

       // Insert the item into the array.
       OrderedItem [] items = {i1};
       po.OrderedItems = items;
       // Calculate the total cost.
       decimal subTotal = new decimal();
       foreach(OrderedItem oi in items)
       {
```

```
            subTotal += oi.LineTotal;
        }
        po.SubTotal = subTotal;
        po.ShipCost = (decimal) 12.51;
        po.TotalCost = po.SubTotal + po.ShipCost;
        // Serialize the purchase order, and close the TextWriter.
        serializer.Serialize(writer, po);
        writer.Close();
    }

    protected void ReadPO(string filename)
    {
        // Create an instance of the XmlSerializer class;
        // specify the type of object to be deserialized.
        XmlSerializer serializer = new XmlSerializer(typeof(Purchase-
Order));
        /* If the XML document has been altered with unknown
        nodes or attributes, handle them with the
        UnknownNode and UnknownAttribute events.*/
        serializer.UnknownNode+= new
        XmlNodeEventHandler(serializer_UnknownNode);
        serializer.UnknownAttribute+= new
        XmlAttributeEventHandler(serializer_UnknownAttribute);

        // A FileStream is needed to read the XML document.
        FileStream fs = new FileStream(filename, FileMode.Open);
        // Declare an object variable of the type to be deserialized.
        PurchaseOrder po;
        /* Use the Deserialize method to restore the object's state with
        data from the XML document. */
        po = (PurchaseOrder) serializer.Deserialize(fs);
        // Read the order date.
        Console.WriteLine ("OrderDate: " + po.OrderDate);

        // Read the shipping address.
        Address shipTo = po.ShipTo;
        ReadAddress(shipTo, "Ship To:");
        // Read the list of ordered items.
        OrderedItem [] items = po.OrderedItems;
        Console.WriteLine("Items to be shipped:");
        foreach(OrderedItem oi in items)
        {
            Console.WriteLine("\t"+
            oi.ItemName + "\t" +
            oi.Description + "\t" +
            oi.UnitPrice + "\t" +
            oi.Quantity + "\t" +
            oi.LineTotal);
        }
        // Read the subtotal, shipping cost, and total cost.
        Console.WriteLine("\t\t\t\t\t Subtotal\t" + po.SubTotal);
```

```
        Console.WriteLine("\t\t\t\t\t Shipping\t" + po.ShipCost);
        Console.WriteLine("\t\t\t\t\t Total\t\t" + po.TotalCost);
    }

    protected void ReadAddress(Address a, string label)
    {
        // Read the fields of the Address object.
        Console.WriteLine(label);
        Console.WriteLine("\t"+ a.Name );
        Console.WriteLine("\t" + a.Line1);
        Console.WriteLine("\t" + a.City);
        Console.WriteLine("\t" + a.State);
        Console.WriteLine("\t" + a.Zip );
        Console.WriteLine();
    }

    private void serializer_UnknownNode
    (object sender, XmlNodeEventArgs e)
    {
        Console.WriteLine("Unknown Node:" +   e.Name + "\t" + e.Text);
    }

    private void serializer_UnknownAttribute
    (object sender, XmlAttributeEventArgs e)
    {
        System.Xml.XmlAttribute attr = e.Attr;
        Console.WriteLine("Unknown attribute " +
        attr.Name + "='" + attr.Value + "'");
    }
}
```

Remarks

XML serialization is the process of converting an object's public properties and fields to a serial format (in this case, XML) for storage or transport. Deserialization re-creates the object in its original state from the XML output. You can think of serialization as a way of saving the state of an object into a stream or buffer. For example, ASP.NET uses the XmlSerializer class to encode XML Web service messages.

The data in your objects is described using programming language constructs like classes, fields, properties, primitive types, arrays, and even embedded XML in the form of XmlElement or XmlAttribute objects. You have the option of creating your own classes, annotated with attributes, or using the XML Schema Definition Tool (Xsd.exe) to generate the classes based on an existing XML Schema definition (XSD) document. If you have an

XML Schema, you can run the Xsd.exe to produce a set of classes that are strongly typed to the schema and annotated with attributes to adhere to the schema when serialized.

To transfer data between objects and XML requires a mapping from the programming language constructs to XML schema and from the XML schema to the programming language constructs. The XmlSerializer and related tools like Xsd.exe provide the bridge between these two technologies at both design time and runtime. At design time, use the Xsd.exe to produce an XML schema document (.xsd) from your custom classes or to produce classes from a given schema. In either case, the classes are annotated with custom attributes to instruct the XmlSerializer how to map between the XML schema system and the common language runtime. At runtime, instances of the classes can be serialized into XML documents that follow the given schema. Likewise, these XML documents can be deserialized into runtime objects. Note that the XML schema is optional, and not required at design time or runtime.

Controlling Generated XML

To control the generated XML, you can apply special attributes to classes and members. For example, to specify a different XML element name, apply an XmlElementAttribute to a public field or property, and set the ElementName property. For a complete list of similar attributes, see Attributes That Control XML Serialization. You can also implement the IXmlSerializable interface to control the XML output.

If the XML generated must conform to section 5 of the World Wide Consortium document, Simple Object Access Protocol (SOAP) 1.1 , you must construct the XmlSerializer with an XmlTypeMapping. To further control the encoded SOAP XML, use the attributes listed in Attributes That Control Encoded SOAP Serialization.

With the XmlSerializer you can take advantage of working with strongly typed classes and still have the flexibility of XML. Using fields or properties of type XmlElement, XmlAttribute or XmlNode in your strongly typed classes, you can read parts of the XML document directly into XML objects.

If you work with extensible XML schemas, you can also use the XmlAnyElementAttribute and XmlAnyAttributeAttribute attributes to serialize and deserialize elements or attributes that are not found in the original schema. To use the objects, apply an XmlAnyElementAttribute to a field that returns an array of XmlElement objects, or apply an XmlAnyAttributeAttribute to a field that returns an array of XmlAttribute objects.

If a property or field returns a complex object (such as an array or a class instance), the XmlSerializer converts it to an element nested within the main XML document. For example, the first class in the following code returns an instance of the second class.

C# Copiar

```
public class MyClass
{
    public MyObject MyObjectProperty;
}
public class MyObject
{
    public string ObjectName;
}
```

The serialized, XML output looks like this:

Copiar

```
<MyClass>
  <MyObjectProperty>
  <ObjectName>My String</ObjectName>
  </MyObjectProperty>
</MyClass>
```

If a schema includes an element that is optional (minOccurs = '0'), or if the schema includes a default value, you have two options. One option is to use System.ComponentModel.DefaultValueAttribute to specify the default value, as shown in the following code.

C# Copiar

```
public class PurchaseOrder
{
    [System.ComponentModel.DefaultValueAttribute ("2002")]
    public string Year;
}
```

Another option is to use a special pattern to create a Boolean field recognized by the XmlSerializer, and to apply the XmlIgnoreAttribute to the field. The pattern is created in the form of `propertyNameSpecified`. For example, if there is a field named "MyFirstName" you would also create a field named "MyFirstNameSpecified" that instructs the XmlSerializer

whether to generate the XML element named "MyFirstName". This is shown in the following example.

C# Copiar

```
public class OptionalOrder
{
    // This field should not be serialized
    // if it is uninitialized.
    public string FirstOrder;

    // Use the XmlIgnoreAttribute to ignore the
    // special field named "FirstOrderSpecified".
    [System.Xml.Serialization.XmlIgnoreAttribute]
    public bool FirstOrderSpecified;
}
```

Overriding Default Serialization

You can also override the serialization of any set of objects and their fields and properties by creating one of the appropriate attributes, and adding it to an instance of the XmlAttributes class. Overriding serialization in this way has two uses: first, you can control and augment the serialization of objects found in a DLL, even if you do not have access to the source; second, you can create one set of serializable classes, but serialize the objects in multiple ways. For more details, see the XmlAttributeOverrides class and How to: Control Serialization of Derived Classes.

To serialize an object, call the Serialize method. To deserialize an object, call the Deserialize method.

To add XML namespaces to an XML document, see XmlSerializerNamespaces.

Observação

The **XmlSerializer** gives special treatment to classes that implement **IEnumerable** or **ICollection**. A class that implements **IEnumerable** must implement a public `Add` method that takes a single parameter. The `Add` method's parameter must be of the same type as is returned from the `Current` property on the value returned from `GetEnumerator`, or one of that type's bases. A class that implements **ICollection** (such as **CollectionBase**) in addition to **IEnumerable** must have a public `Item` indexed property (indexer in C#) that takes an integer, and it must have a public `Count`

property of type integer. The parameter to the `Add` method must be the same type as is returned from the `Item` property, or one of that type's bases. For classes that implement **ICollection**, values to be serialized are retrieved from the indexed `Item` property, not by calling `GetEnumerator`.

You must have permission to write to the temporary directory (as defined by the TEMP environment variable) to deserialize an object.

Dynamically Generated Assemblies

To increase performance, the XML serialization infrastructure dynamically generates assemblies to serialize and deserialize specified types. The infrastructure finds and reuses those assemblies. This behavior occurs only when using the following constructors:

XmlSerializer.XmlSerializer(Type)

XmlSerializer.XmlSerializer(Type, String)

If you use any of the other constructors, multiple versions of the same assembly are generated and never unloaded, which results in a memory leak and poor performance. The easiest solution is to use one of the previously mentioned two constructors. Otherwise, you must cache the assemblies in a Hashtable, as shown in the following example.

C# Copiar

```
Hashtable serializers = new Hashtable();

// Use the constructor that takes a type and XmlRootAttribute.
XmlSerializer s = new XmlSerializer(typeof(MyClass), myRoot);

// Implement a method named GenerateKey that creates unique keys
// for each instance of the XmlSerializer. The code should take
// into account all parameters passed to the XmlSerializer
// constructor.
object key = GenerateKey(typeof(MyClass), myRoot);

// Check the local cache for a matching serializer.
XmlSerializer ser = (XmlSerializer)serializers[key];
if (ser == null)
{
    ser = new XmlSerializer(typeof(MyClass), myRoot);
    // Cache the serializer.
    serializers[key] = ser;
}
```

```
// Use the serializer to serialize or deserialize.
```

Serialization of ArrayList and Generic List

The XmlSerializer cannot serialize or deserialize the following:

- Arrays of ArrayList
- Arrays of List<T>

Serialization of Enumerations of Unsigned Long

The XmlSerializer cannot be instantiated to serialize an enumeration if the following conditions are true: The enumeration is of type unsigned long (`ulong` in C#) and the enumeration contains any member with a value larger than 9,223,372,036,854,775,807. For example, the following cannot be serialized.

Copiar

```
public enum LargeNumbers: ulong
{
    a = 9223372036854775808
}
// At runtime, the following code will fail.
xmlSerializer mySerializer=new XmlSerializer(typeof(LargeNumbers));
```

Objects marked with the Obsolete Attribute no longer serialized

In the .NET Framework 3.5 the XmlSerializer class no longer serializes objects that are marked as `[Obsolete]`.

Constructors

XmlSerializer()	Initializes a new instance of the XmlSerializer class.
XmlSerializer(Type)	Initializes a new instance of the XmlSerializer class that can serialize objects of the specified type into XML documents, and

	deserialize XML documents into objects of the specified type.
XmlSerializer(Type, String)	Initializes a new instance of the XmlSerializer class that can serialize objects of the specified type into XML documents, and deserialize XML documents into objects of the specified type. Specifies the default namespace for all the XML elements.
XmlSerializer(Type, Type[])	Initializes a new instance of the XmlSerializer class that can serialize objects of the specified type into XML documents, and deserialize XML documents into object of a specified type. If a property or field returns an array, the `extraTypes` parameter specifies objects that can be inserted into the array.
XmlSerializer(Type, XmlAttributeOverrides)	Initializes a new instance of the XmlSerializer class that can serialize objects of the specified type into XML documents, and deserialize XML documents into objects of the specified type. Each object to be serialized can itself contain instances of classes, which this overload can override with other classes.
XmlSerializer(Type, XmlAttributeOverrides, Type[], XmlRootAttribute, String)	Initializes a new instance of the XmlSerializer class that can serialize objects of type Object into XML document instances, and deserialize XML document instances into objects of type Object. Each object to be serialized can itself contain instances of classes, which this overload overrides with other classes. This overload also specifies the default namespace for all the XML elements and the class to use as the XML root element.
XmlSerializer(Type, XmlAttributeOverrides, Type[], XmlRootAttribute, String, String)	Initializes a new instance of the XmlSerializer class that can serialize objects of type Object into XML document instances, and deserialize XML document instances into objects of type Object. Each object to be serialized can itself contain instances of classes, which this overload overrides with other classes. This overload also specifies the default namespace for all the XML elements and the class to use as the XML root element.
XmlSerializer(Type, XmlRootAttribute)	Initializes a new instance of the XmlSerializer class that can serialize objects of the specified type into XML documents, and deserialize an XML document into object of the specified type. It also specifies the class to use as the XML root element.
XmlSerializer(XmlTypeMapping)	Initializes an instance of the XmlSerializer class using an object that maps one type to another.

Methods

CanDeserialize(XmlReader)	Gets a value that indicates whether this XmlSerializer can deserialize a specified XML document.
CreateReader()	Returns an object used to read the XML document to be serialized.
CreateWriter()	When overridden in a derived class, returns a writer used to serialize the object.
Deserialize(Stream)	Deserializes the XML document contained by the specified Stream.
Deserialize(TextReader)	Deserializes the XML document contained by the specified TextReader.
Deserialize(XmlReader)	Deserializes the XML document contained by the specified XmlReader.
Deserialize(XmlReader, String)	Deserializes the XML document contained by the specified XmlReader and encoding style.
Deserialize(XmlReader, String, XmlDeserializationEvents)	Deserializes the object using the data contained by the specified XmlReader.
Deserialize(XmlReader, Xml DeserializationEvents)	Deserializes an XML document contained by the specified XmlReader and allows the overriding of events that occur during deserialization.
Deserialize(XmlSerialization Reader)	Deserializes the XML document contained by the specified XmlSerializationReader.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
FromMappings(XmlMapping[])	Returns an array of XmlSerializer objects created from an array of XmlTypeMapping objects.
FromMappings(XmlMapping[], Type)	Returns an instance of the XmlSerializer class from the specified mappings.

FromTypes(Type[])	Returns an array of XmlSerializer objects created from an array of types.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
GetXmlSerializerAssemblyName(Type)	Returns the name of the assembly that contains one or more versions of the XmlSerializer especially created to serialize or deserialize the specified type.
GetXmlSerializerAssemblyName(Type, String)	Returns the name of the assembly that contains the serializer for the specified type in the specified namespace.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Serialize(Object, XmlSerializationWriter)	Serializes the specified Object and writes the XML document to a file using the specified XmlSerializationWriter.
Serialize(Stream, Object)	Serializes the specified Object and writes the XML document to a file using the specified Stream.
Serialize(Stream, Object, XmlSerializerNamespaces)	Serializes the specified Object and writes the XML document to a file using the specified Stream that references the specified namespaces.
Serialize(TextWriter, Object)	Serializes the specified Object and writes the XML document to a file using the specified TextWriter.
Serialize(TextWriter, Object, XmlSerializerNamespaces)	Serializes the specified Object and writes the XML document to a file using the specified TextWriter and references the specified namespaces.
Serialize(XmlWriter, Object)	Serializes the specified Object and writes the XML document to a file using the specified XmlWriter.
Serialize(XmlWriter, Object, XmlSerializerNamespaces)	Serializes the specified Object and writes the XML document to a file using the specified XmlWriter and references the specified namespaces.
Serialize(XmlWriter, Object, Xml	Serializes the specified object and writes the XML document to a

SerializerNamespaces, String)	file using the specified XmlWriter and references the specified namespaces and encoding style.
Serialize(XmlWriter, Object, XmlSerializerNamespaces, String, String)	Serializes the specified Object and writes the XML document to a file using the specified XmlWriter, XML namespaces, and encoding.
ToString()	Returns a string that represents the current object. (Inherited from Object)

Events

UnknownAttribute	Occurs when the XmlSerializer encounters an XML attribute of unknown type during deserialization.
UnknownElement	Occurs when the XmlSerializer encounters an XML element of unknown type during deserialization.
UnknownNode	Occurs when the XmlSerializer encounters an XML node of unknown type during deserialization.
UnreferencedObject	Occurs during deserialization of a SOAP-encoded XML stream, when the XmlSerializer encounters a recognized type that is not used or is unreferenced.

Applies to

Produto	Versões
.NET	5.0
.NET Core	1.0, 1.1, 2.0, 2.1, 2.2, 3.0, 3.1
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8
.NET Standard	2.0, 2.1
UWP	10.0
Xamarin.Android	7.1

Produto	Versões
Xamarin.iOS	10.8
Xamarin.Mac	3.0

Thread Safety

This type is thread safe.

See also

- XmlAttributeOverrides
- XmlAttributes
- XmlSerializer
- XmlText
- Introducing XML Serialization
- How to: Specify an Alternate Element Name for an XML Stream
- Controlling XML Serialization Using Attributes
- Examples of XML Serialization
- XML Schema Definition Tool (Xsd.exe)
- How to: Control Serialization of Derived Classes
- <dateTimeSerialization>Element
- <xmlSerializer> Element

Esta página é útil?

Yes No